Amélioration du positionnement de fragments ADN pour réalisation de séquences consensus dans le contexte de l'assemblage de novo de longues lectures

Victor Epain

Université de Rennes 1
UFR Sciences de la Vie
et de l'Environnement

Amélioration du positionnement de fragments ADN pour réalisation de séquences consensus dans le contexte de l'assemblage *de novo* de longues lectures

Master 2 Bioinformatique
— Rapport de stage —
Soutenance : 16 juin 2020

Victor Epain

Laboratoire de recherche Inria

RBA – Campus de Beaulieu

35042 Rennes CEDEX

Équipe GenScale

Sous la responsabilité de :
Rumen Andonov, Pr Univ. Rennes 1
Dominique Lavenier, DR cnrs

3 février 2020 — 17 juillet 2020

Master bioinformatique

UNIVERSITÉ DE RENNES 1

ENGAGEMENT DE NON PLAGIAT

Je, soussigné(e) ..EPAIN Victor.....
étudiant(e) en ..Master 2 de Bioinformatique, Université de Rennes 1.....
déclare être pleinement informé que le plagiat de documents ou d'une partie de document publiés sur toute forme de support, y comprit l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour la rédaction de ce document.

Date : Le 13 mai 2020, à Rennes

Signature :

Document à compléter de manière manuscrite et à insérer obligatoirement en première page du rapport de stage.

Avant propos et remerciements

Avant-propos

Pour éviter toute ambiguïté liée à mon expression en langue française, certaines formules mathématiques ont été décrites aux côtés des textes associés. Toutefois, la description en français suffit à la compréhension.

Remerciements

Je souhaiterais remercier M. Andonov et M. Lavenier de me permettre de continuer nos recherches entamées l'année dernière sur la problématique de l'assemblage *de novo* de longues lectures par l'outil mathématique. Aussi, de m'avoir offert la chance de présenter nos travaux au congrès national sur la recherche opérationnelle et ses applications ROADEF 2020 à Montpellier fin février. Je remercie également Jean-François Gibrat avec qui nous avons partagé nos méthodes pour l'assemblage de génome lors de sa visite au laboratoire IRISA / Inria.

Aussi, je remercie Émeline Roux pour m'avoir donné accès à un génome de bactérie séquencé par une technologie de séquençage longues lectures, et pour m'avoir conseillé sur les génomes bactériens à utiliser en jeux de tests.

Enfin, je souhaiterais remercier Marie Le Roïc qui m'a accompagné dans les démarches administratives, en particulier celles concernant la visite à Montpellier.

Liste des abréviations et symboles

Abréviations

Abau Acinetobacter baumannii

B1NL Bacillus sp. 1NLA3E

Ecol Escherichia coli

EF Erreurs Fortes

Hpyl Helicobacter pylori

LCM Longueur de Chevauchement Minimale

Mbov Mycobacterium bovis

MTZ Contrainte Miller-Tucker-Zemlin

NA Nombre d'Arcs

NCBI National Center for Biotechnology Information

NN Nombre de Nœuds

PE Pseudos-Erreurs

PLMNE Programmation mathématique Linéaire Mixte en Nombres Entiers

Pmit Prochlorococcus sp. MIT 0604

Saur Staphylococcus aureus

Sent Salmonella enterica

Sthe Streptococcus thermophilus

Ypes Yersinia pestis

Table des matières

Chapitre 1 : Introduction

1.1 L'assemblage mono-molécule ADN *de novo*

L'analyse *in silico* de la structure génomique d'une seule molécule ADN d'un organisme, correspondant à un chromosome ou à un plasmide chez une bactérie, dépend de deux étapes préliminaires : obtenir des fragments ADN, appelés lectures (*reads* dans la littérature anglophone), en séquençant la molécule, puis assembler ces lectures en espérant n'obtenir qu'une seule séquence correspondant à la molécule de départ. Lors du séquençage, les deux brins d'ADN sont indifférenciés, et les lectures d'un brin sont, en séquence, inversées et complémentaires par rapport à celles du brin complémentaire. L'assemblage *de novo* est une méthode d'assemblage de lectures ADN qui ne repose sur aucune comparaison avec une référence - celle ci pouvant être la molécule ADN analogue chez un organisme de la même espèce ou d'une espèce proche.

Aujourd'hui, les nouvelles technologies de séquençage produisent un très grand nombre de lectures, car si l'on calcule la profondeur moyenne de séquençage, c'est à dire le nombre de nucléotides qui s'alignent sur le génome d'origine divisé par la taille du génome, on peut atteindre un facteur de l'ordre de la centaine. Toutefois, ces technologies ne garantissent pas une fiabilité de leurs séquences nucléotidiques à 100%, et plusieurs types d'erreurs de séquençage interviennent :

— des substitutions : des nucléotides sont changés en un autre

— des insertions : un ou plusieurs nucléotides sont de trop par rapport à ce qu'aurait dû être la séquence de la lecture issue du génome d'origine

— des suppressions : analogue au point précédent, mais pour un manque d'un ou plusieurs nucléotides

Nommés *indels* dans la littérature anglophone, ces deux derniers types d'erreurs seront indifférenciés et désignés par *insup* pour la suite.

L'abondance de lectures offerte par le séquençage est un avantage informationnel. En effet, on espère qu'une grande profondeur de séquençage peut corriger les erreurs, le paradigme sous-entendu étant celui du consensus : bien que la méthode associée reste à déterminer, l'idée principale est celle d'un vote majoritaire qui décide la forme finale d'une unique séquence à partir de plusieurs - à savoir celles de lectures similaires.

L'avantage informationnel est balancé d'un inconvénient de taille en mémoire : au mieux, deux bits suffisent pour décrire les quatre nucléotides, mais en général, les nucléotides étant représentés par les lettres A, T, G, C, ils sont encodés en ASCII *i.e.* sur 8 bits. Pour ne prendre que cet exemple, le génome de la bactérie *Escherichia coli str. K-12 substr. MG1655* présente sur le site du *National Center for Biotechnology Information* (NCBI) est long de 4 639 694 nucléotides (paires de bases,

car l'ADN est double brins). Multiplié par une profondeur de 100, le stockage des lectures coûte en mémoire $100 * 4,6 * 10^6 * 8 = 3680Mo$ soit $3,59Go$. Et ce, sans compter le poids de l'information issue de leurs comparaisons.

En outre, le nombre seul de lectures ne garantit pas un succès d'assemblage : la taille de ces lectures est un facteur de faisabilité et de succès. Les génomes possèdent des régions répétées : pour les bactéries, ces régions varient en longueur du millier de paires de bases à de petites répétitions en tandem long.[1] Ainsi, de longues lectures peuvent couvrir les régions répétées, permettant ainsi leur différenciation grâce aux régions flanquantes. Alors que la technologie de séquençage Illumina[2] génère des courtes lectures de taille de l'ordre de la centaine de nucléotides, les technologies de séquençage Oxford Nanopore avec la portabilité du Minion[3] ou encore la technologie PacBio[4] en génèrent des longues, variant de plusieurs centaines à la centaine de millier. Cependant, les erreurs de séquençage de ces deux dernières technologies sont plus nombreuses et sont davantage des *insup*, ce qui rend la problématique de l'assemblage bien plus ardue, car provoque des décalages nucléotidiques. En comparaison, les rares erreurs de séquençage courtes lectures sont des substitutions.

Bien qu'il existe des méthodes d'assemblages hybrides, c'est à dire profitant de la taille des longues lectures pour surpasser l'issue des régions répétées et le faible taux d'erreurs des courtes lectures pour améliorer la qualité d'une séquence résultante d'un consensus, nous nous pencherons sur l'assemblage à partir uniquement de longues lectures.

1.2 Les méthodes actuelles d'assemblages *de novo* longues lectures mono-molécule ADN

Les différentes méthodes d'assemblage reposent sur des modélisations des données de départ différentes. Pour exemples, l'assembleur Canu[5] utilise la comparaison de chevauchements des lectures. Il choisit les lectures en fonction d'un score de qualité et corrige en amont les séquences. Une séparation de ces lectures est effectuée lorsqu'elles sont considérées dans des régions de résolution difficiles. Enfin, les lectures sont assemblées en détectant les erreurs en leur sein pour former des contigs issus d'un consensus.

L'assembleur wtdbg2[6] compare les lectures en les alignant à partir d'un découpage de leurs séquences par k-mers (un k-mer est une petite séquence de taille k). Un graphe de régions des lectures alignées est ensuite construit, où chaque nœud représente un ensemble de régions de plusieurs lectures alignées. Ces nœuds sont reliés quand les régions d'un ensemble étaient consécutives aux régions d'un autre ensemble. Ce graphe est nettoyé jusqu'à n'avoir qu'un seul chemin. Finalement, à partir des nœuds restant et des liaisons entre ces nœuds, une séquence unique issue d'un consensus dans chaque ensemble est réalisée.

Pour finir, l'assembleur Fly[7] génère à partir d'un graphe d'alignements de lectures des séquences

plus longues (nommées *contigs*) disjoints dans le graphe. Ils sont ensuite concaténés et la concaténation est comparée avec elle même sur un *dot-plot* qui montre les régions de cette concaténation qui sont répétées. Un graphe de répétition est ainsi construit et les liaisons entre régions répétées (qui représentent des ambiguïtés de chemins dans ce graphe de répétitions) sont tranchées grâce à l'alignement des lectures sur ce graphe de répétitions. Cette approche est une attaque directe contre les régions répétées dans les génomes.

1.3 LOREAS, une nouvelle stratégie d'assemblage *de novo* de longues lectures mono-molécule ADN

Alors que la stratégie de CANU repose sur une correction accrue des lectures pour profiter pleinement de l'avantage des tailles des longues lectures - que celle de WTDBG2 s'appuie sur l'information donnée par les k-mers de lectures qui s'alignent pour nettoyer son graphe d'assemblage et ensuite corriger les erreurs de séquençage - et qu'enfin, FLY s'abstrait des erreurs pour résoudre la difficulté des répétitions dans les génomes ; nous proposons une nouvelle stratégie que nous avons nommée LOREAS (pour LOng REads ASsembler), projet qui a débuté en avril 2019, avec le souhait de formaliser la problématique de l'assemblage par un ensemble de sous-problèmes se résolvant à l'aide de la programmation mathématique linéaire mixte en nombres entiers (PLMNE).

À partir des lectures et des chevauchements entre elles, un graphe de chevauchements est construit. Ensuite, il s'agit de positionner le maximum de lectures sur un axe objectif (dans le sens d'unique) de positions. Cette étape se réalise avec la PLMNE. Ensuite, l'axe objectif des positions est découpé par fenêtres également par PLMNE. Enfin, dans chaque fenêtre, une séquence consensus est construite à partir des régions des lectures s'y trouvant, soit en s'appuyant sur la PLMNE (mentionné), soit par alignement multiple.

Cette stratégie d'assemblage est non spécialisée, et a été testée sur des petits génomes (taille de l'ordre du million de nucléotides) en considérant qu'ils étaient linéaires.

Ce rapport mettra l'accent sur la problématique du positionnement pour un maximum de lectures afin de garantir une force informationnelle pour réaliser le consensus, et ainsi obtenir un assemblage de bonne qualité.

Chapitre 2 : Matériel

2.1 Langage de programmation et matériel physique

AMPL 3.5.0[8] un langage d'implémentation mathématique : AMPL est un language de programmation mathématique commercial. Tous les modèles mathématiques en PLMNE ont été implémentés avec.

Programmation avec Python 3.8 et Linux Bash 5.0.16 : Tous les programmes ont été écrits sous le langage de programmation Python 3.8. Aussi, les modules utilisés sont les suivants : NetworkX 2.4[9] pour manipuler les graphes ; Matplotlib 3.2.1[10] pour la génération automatique des figures ; NumPy 1.18.0[11] et SciPy 1.4.1[12] pour les calculs statistiques et l'usage des objets matriciels. Les quelques commandes pour lancer les programmes ont été écrites en Bash 5.0.16.

GEPHI 0.9.2[13] : GEPHI est un outil open-source de visualisation et de calculs statistiques de graphes.

Matériel physique : Les programmes ont principalement tournés sur un ordinateur personnel sous le système d'exploitation UBUNTU 20.04. L'ordinateur portable disposait de 16 GB RAM, processeur Intel i5 7^{e} génération, CPU 2.50GHz et un GPU NVIDIA GeForce GTX 1050 - 2 GB RAM. Certains scripts ont été exécutés sur le cluster de calcul et de stockage GenOuest, et ont profité de la capacité en RAM et en threads.

2.2 Présentation de la PLMNE

La programmation mathématique est l'implémentation d'une fonction objective dont on souhaite maximiser ou minimiser la valeur, et ce, sous contraintes écrites sous la forme d'équations. Un solveur doit ensuite résoudre le problème énoncé. C'est une méthode qui, quand le problème possède une solution, renvoie une solution exacte. Il est donc intéressant de bien énoncer le problème en choisissant la bonne fonction objective et écrire les bonnes contraintes.

Linéaire signifie que toutes les équations sont écrites sous la forme d'une combinaison linéaire des variables. Par exemple :

$$c_1 * x_1 + c_2 * x_2 + c_3 * x_3 \leq B$$

où x_i, $i \in \{1, 2, 3\}$ sont des variables, c_i, $i \in \{1, 2, 3\}$ et B des constantes.

La méthode de résolution utilisée par le solveur est la méthode du *Branch and Bound*[14] originellement proposée par Land et Doig : elle explore l'arbre combinatoire du possible en comparant les bornes inférieure et supérieure à la valeur actuelle de la fonction objective, en fonction des branches

par lesquels l'algorithme est passé. Cette comparaison permet d'orienter la recherche dans l'arbre en éliminant des nœuds ou en orientant les priorités d'explorations. La recherche d'une valeur optimale peut toutefois coûter cher en temps, et la combinatoire des problèmes peut être un inconvénient en mémoire.

GUROBI 8.1.0[15] un solveur associé à l'implémentation en AMPL : GUROBI est un solveur qui peut résoudre les PLMNE.

2.3 Programmes et logiciels informatiques

MINIMAP2 2.17[16] pour l'alignement de longues lectures : MINIMAP2 est un aligneur de séquences nucléotidiques spécialisé pour les longues lectures victimes d'un fort taux d'erreurs. Il peut aussi bien réaliser un alignement d'un ensemble de séquences sur une référence qu'aligner toutes *versus* toutes. L'aligneur propose aussi des ensembles de paramètres en fonction de la nature des séquences *i.e.* si elles sont issues de la technologie Nanopore ou PacBio.

BLAST 2.10.0+[17] - un aligneur de séquences : BLAST est un aligneur de séquences heuristique, en particulier nucléotidiques avec la suite blastn.

METIS 5.1.0[18] - un partitionneur de graphe : METIS est un partitionneur multi-niveaux de graphes non orientés, utile au partitionnement du graphe de chevauchements d'origine, *cf.* figure (3.4). Bien que son usage ne sera pas décrit dans ce rapport, il semblait toutefois correct d'en faire mention, par son importance en soi dans la stratégie, traitée simplement dans un précédent rapport de stage.[19]

DEEPSIMULATOR 1.5[20] - un simulateur de séquençage Oxford Nanopore : DEEPSIMULATOR est un programme open source qui génère en utilisant du deep-learning des longues lectures avec un taux d'erreurs semblable à celui en sortie de séquençage Oxford Nanopore. Plus précisément, il simule le résultat en sortie de séquenceur et base-calling de la technologie. Toutefois, la distribution des tailles des lectures est très homogène par rapport à une sortie réelle du séquenceur.

MUSCLE 3.8.31[21] - un aligneur multiple de séquences : MUSCLE est un aligneur multiple de séquences, originellement protéiques, mais maintenant aussi nucléotidiques. Bien qu'il existe des algorithmes en temps polynomial pour chercher l'alignement optimale entre deux séquences, la généralisation à k séquences est NP-complet - quand il s'agit de maximiser la somme des scores des pairs de chaque alignements locaux deux séquences par deux.[22]

2.4 Données génomiques

Le tableau suivant montre les génomes de bactéries utilisés pour tester la stratégie, et qui apparaissent dans les résultats. Tous sont extraits de la base de données du NCBI :

Bactéries	NCBI database ID	Taille (en pb)
Acinetobacter baumannii (Abau)	NZ_CP009257.1	4 335 793
Bacillus sp. 1NLA3E (B1NL)	NC_021171.1	4 815 602
Escherichia coli (Ecol)	NZ_CP032667.1	4 639 694
Helicobacter pylori (Hpyl)	NC_000915.1	1 667 867
Mycobacterium bovis (Mbov)	NC_002945.4	4 349 904
Prochlorococcus sp. MIT 0604 (Pmit)	NZ_CP007753.1	1 780 061
Salmonella enterica (Sent)	NC_003198.1	4 809 037
Staphylococcus aureus (Saur)	NC_007795.1	2 821 361
Streptococcus thermophilus (Sthe)	NC_017581.1	1 929 905
Yersinia pestis (Ypes)	NC_003143.1	4 653 728

Table 2.1 – La première colonne représente le nom de la bactérie et le code raccourcis associé, la seconde donne l'identifiant du génome dans la base de donnée du NCBI et la dernière donne la taille de ce génome en paires de bases.

Ce benchmark a été proposé par Émeline Roux, experte pour les génomes bactériens, car ces bactéries couvrent des familles bactériennes différentes et certaines possèdent beaucoup de régions répétées. À l'exception du fichier de longues lectures de Sthe, issu d'un véritable séquençage Minion et de profondeur moyenne égale à $100X$, tous les autres ont été générés artificiellement avec le générateur de longues lectures DeepSimulator, avec une profondeur de séquençage simulée de $30X$. Bien que tous ces génomes soient circulaires, les génomes simulés séquencés l'ont été en omettant cette caractéristique, et en les considérant linéaires.

Chapitre 3 : Méthodes

3.1 Contexte de résolution du problème : milieu de la phase de positionnement

3.1.1 Données et modélisation de départ

Alignement et définitions des chevauchements entre lectures : Les lectures sont alignées les unes contre les autres, dans le sens direct et inverse-complémentaire, avec MINIMAP2 et l'option **-x ava-ont** spécifique pour le séquençage Nanopore. Les alignements passent ensuite un filtre, qui ne garde au plus qu'un seul alignement entre une lecture u et une lecture v. Seuls les chevauchements *i.e.* les alignements de types suffix-préfixe, et de longueur au dessus d'un seuil λ_{min} ont été gardés ; *cf.* sous-figure (3.2a). D'autres heuristiques de filtre ne seront pas précisées ici (*cf.* rapport de stage précédent[19]).

Définition 3.1-1 : Attributs d'un chevauchement

Soit la figure (3.1) représentant deux lectures u et v, de tailles respectives l_u et l_v, alignées par MINIMAP2. Le chevauchement $(u,\ v)$ est défini par :

longueur $\lambda_{uv} = max(al_u,\ al_v) + v_1 + (l_u - u_2)$

gap $g_{uv} = max(0,\ al_u - al_v)$

où $al_u = u_2 - u_1,\ al_v = v_2 - v_1$

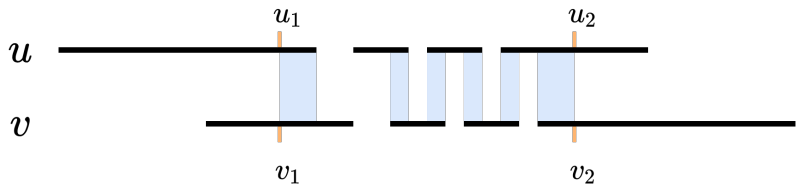
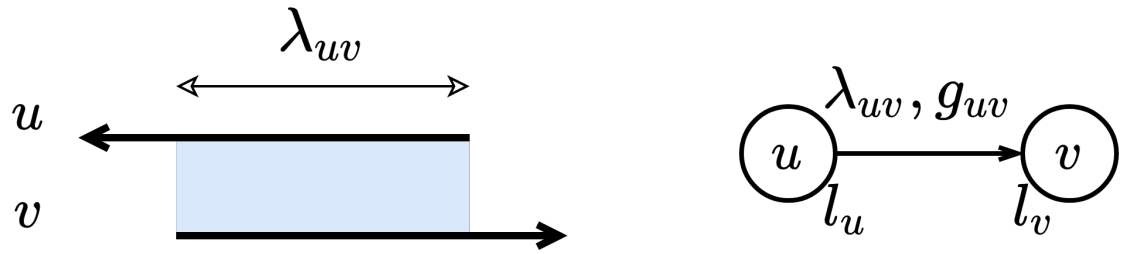


Figure 3.1 – u et v sont deux lectures qui ont été alignées par Minimap2. Les zones bleues représentent les régions d'identités et de substitutions entre les deux séquences. Elles sont séparées de zones blanches qui sont les conséquences d'un *insup* sur la continuité de l'alignement, à la fois sur u, et sur v. u_1, u_2 sont les coordonnées de départ et de fin d'alignement sur u, *mutadis mutandis* pour v.

Construction du graphe de chevauchements :



(a) Vision simplifiée d'un chevauchement entre une lecture u et v

(b) Représentation d'un chevauchement u vers v par un graphe

FIGURE 3.2 – Deux lectures u et v, de longueurs respectives l_u et l_v qui se chevauchent.
(3.2a) : La séquence inverse complémentaire de u chevauche la séquence en sens direct v, avec une longueur λ_{uv} selon la définition 3.1-1.
(3.2b) : Bien que les séquences qui se chevauchent soient orientées, pour ne pas alourdir la représentation, les orientations ne sont pas indiquées. En réalité, pour cette représentation, $(u_r, v_f) \in E$; où f, r sont respectivement les sens directs ($forward$) et inverse-complémentaire ($reverse$). Dans le graphe, on a donc aussi le réciproque $(v_r, u_f) \in E$ - et les longueurs de chevauchement λ_{uv} pour ce couple et son réciproque sont les mêmes.

Soit le graphe orienté de chevauchements $OG = (V, E, l_V, \lambda_E, g_E)$, où V (pour $Vertices$ en anglais) est l'ensemble des sommets du graphe qui sont les lectures de tailles dans la liste l_V, E (pour $Edges$) l'ensemble des arcs $(u, v) \in V^2$ qui sont les chevauchements d'une lecture u vers une lecture v, pondérés par la longueur de chevauchement dans la liste λ_E et la différence de longueur d'alignement sur u et v dans la liste g_E ; $cf.$ sous figure (3.2b). Il est important de noter que si les deux sens d'une lecture v participe dans les arcs, alors elles sont différenciées avec deux nœuds v_f et v_r.

3.1.2 Réduction de la taille de l'instance à traiter : partitionnement du graphe OG

Axiome 3.1-1 : Partitionnement du graphe de chevauchements d'origine

(i) Le graphe de chevauchements principal OG a été partitionné en n_{parts} sous-graphes de chevauchements $OG_i,\ i \in [\![1\,;\, n_{parts}]\!]$ correspondant chacun à une région du génome de profondeur locale X_i ;

(ii) l'ordre de résolution de ces graphes de chevauchements a été donné dans le bon ordre ;

(iii) chaque sous-graphe OG_i n'a qu'une seule composante connexe faible.

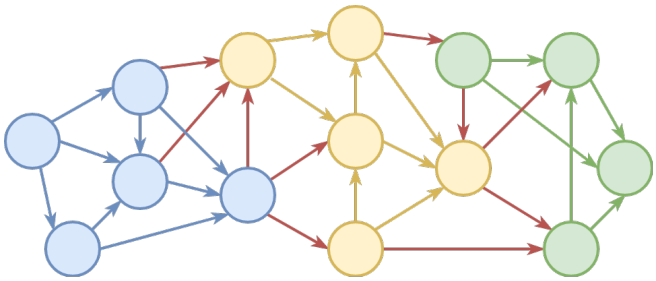


Figure 3.3 – Illustration d'un graphe de chevauchements partitionné en trois sous-graphes (bleu-jaune-vert). Les arcs en rouge représentent les liaisons entre ces sous-graphes OG_i.

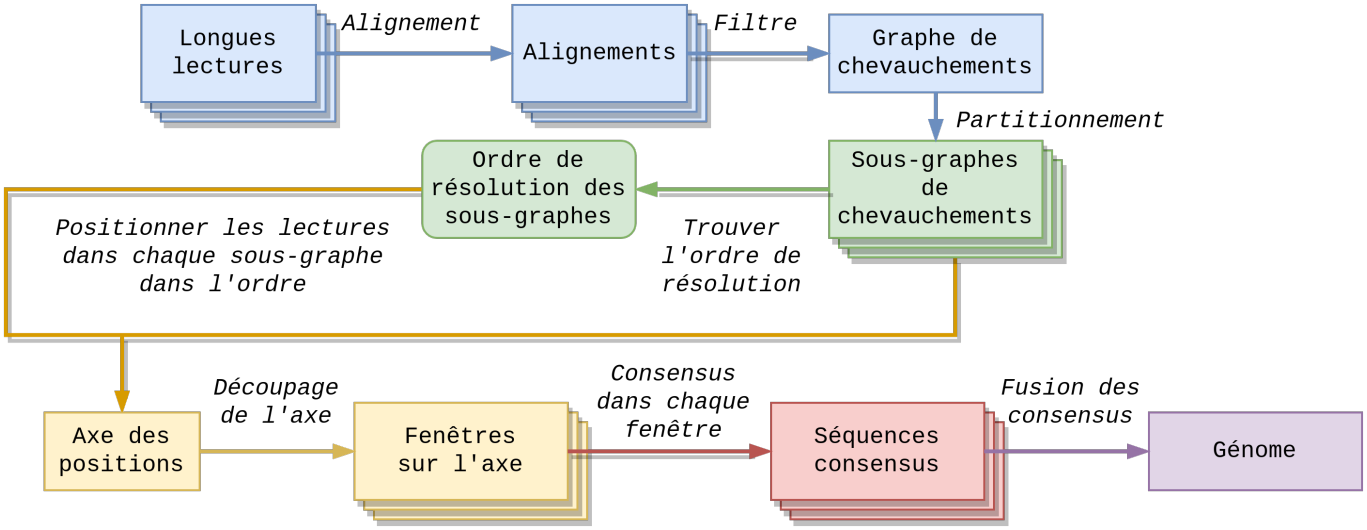


Figure 3.4 – Vue d'ensemble de la stratégie Loreas. La présentation démarre juste à la fin des étapes vertes.

Il s'agit de trouver au moins un chemin élémentaire par sous-graphe de chevauchements OG_i rejoignant le ou les chemin(s) des sous-graphes adjacents. Cela revient à positionner les lectures localement pour chaque région du génome, en vertu du point (i) de l'axiome 3.1-1.

Une fois tous les sous-graphes de chevauchements résolus pour ce problème, on obtient pour chaque lecture dans les chemins une position, ce qui nous permet de les positionner sur un seul axe de position *pos*. On souhaite ensuite réaliser une séquence consensus en profitant de ce positionnement. Pour cela, nous proposons une découpe en fenêtre de cet axe *pos* et la réalisation dans chaque fenêtre d'une séquence consensus (où deux méthodes sont présentées). Une fois que l'on a une séquence consensus par fenêtre, on procède à leur fusion pour n'obtenir *in fine* qu'une seule séquence consensus, dont on souhaite qu'elle représente le génome.

3.2 Recherche de multi-chemins dans un graphe de chevauchements $OG_i = (V_i, E_i, l_{V_i}, \lambda_{E_i}, g_{E_i})$

Précédemment, nous ne cherchions à construire qu'un seul chemin élémentaire pour chaque sous-graphe et à concaténer tous les chemins. Dans cette section, nous recherchons dans chaque sous-graphe de chevauchements un grand nombre de chemins, ce qui revient à exploiter la profondeur de séquençage. Soit un graphe de chevauchement $OG_i = (V_i, E_i, l_{V_i}, \lambda_{E_i}, g_{E_i}),\ i \in [\![1\,;\, n_{parts}]\!]$ où n_{parts} est le nombre de sous-graphes.

Définition 3.2-1 : Données pour la PLMNE

— R_i : les couples des lectures direct / inverse-complémentaire de la forme $(v_f, v_r) \in V_i^2$;

— $M = \sum_{v \in V_i} l_v$, *big M* dans la littérature, une constante borne supérieure.

Les modélisations en PLMNE suivantes s'appuient largement sur celle proposée par S. François, R. Andonov et al.[23]

3.2.1 Recherche en séquentielle

Définition 3.2-2 : Variables pour la PLMNE

— $x_{uv}, \forall\, (u,v) \in E_i$, où x_{uv} vaut 1 si l'arc (u,v) participe au chemin, 0 sinon ;

— $y_v \in \mathbb{N},\ \forall\, v \in V_i$ la position la plus à droite de la lecture v. Vaut 0 si le nœud associé dans le graphe ne participe pas à un chemin ;

— $\forall\ v \in V_i,\ s_v + i_v + t_v \leq 1$ des variables binaires qui valent 1 si, respectivement, v est débutant, intermédiaire ou terminal dans le chemin, sinon 0.

Postulat 3.2-1 : Fonction objective

On admet que la reconstitution de la région locale du génome associée à ce sous-graphe correspond à trouver le plus lourd chemin, selon les longueurs de chevauchements λ_{E_i}, dans ce graphe. La fonction objective associée est : $\quad maximizer \sum_{(u,v) \in E_i} \lambda_{uv} \times x_{uv}$

Définition 3.2-3 : Contraintes pour la PLMNE

Définition d'un chemin élémentaire :

— $\sum_{v \in V_i} s_v = 1 \ \sum_{v \in V_i} t_v = 1$: un chemin n'a qu'un seul débutant et qu'un seul terminal ;

— $\forall\ v \in V_i,\ \sum_{(v,u)\in E_i} x_{vu} = s_v + i_v$: un nœud débutant ou intermédiaire a un successeur dans le chemin ;

— $\forall\ v \in V_i,\ \sum_{(u,v)\in E_i} x_{uv} = i_v + t_v$: un nœud intermédiaire ou terminal a un prédécesseur dans le chemin.

Contraintes de positionnement :

— $\forall\ v \in V,\ y_v \geq s_v \times l_v$: initialise la valeur de y pour la première lecture ;

— Si le chevauchement (u, v) est choisi, alors les positions en tête des lectures u et v doivent respecter les attributs du chevauchement :

$$\forall\ (u,v) \in E : \begin{cases} y_v \geq y_u + (l_v - (\lambda_{uv} - g_{uv}))x_{uv} - (1 - x_{uv})M \\ y_v \leq y_u + (l_v - (\lambda_{uv} - g_{uv}))x_{uv} + (1 - x_{uv})M \end{cases}$$

contrainte type Miller-Tucker-Zemlin[24] (MTZ) pour prévenir la formation de boucles dans le chemin.

Contrainte unicité de sens des lectures :

— $\forall\ (v_f, v_r) \in R_i,\ s_{v_f} + i_{v_f} + t_{v_f} + s_{v_r} + i_{v_r} + t_{v_r} \leq 1$, *i.e.* si la lecture est positionnée, alors elle l'est en séquence direct sinon inverse-complémentaire.

Ainsi, on va rechercher le plus lourd chemin dans ce graphe, puis rechercher à nouveau le plus lourd chemin dans le même graphe privé des nœuds participant aux chemins trouvés aux itérations précédentes, etc. La recherche s'arrête lorsqu'il n'est plus possible de trouver un nouveau chemin où que celui trouvé ne soit pas assez lourd par rapport à la taille nucléotidique qu'il représente ou bien que cette taille soit trop faible par rapport à la taille du premier chemin trouvé. Le critère d'arrêt est énoncé mathématiquement en annexe A.1, postulat A.1-1.

3.2.2 Recherche en parallèle

À la différence de la recherche séquentielle, le maximum (au plus $n_{paths} \in \mathbb{N}^*$), plus lourds chemins parallèles seront recherchés en une seule itération dans ce graphe OG_i. Cette fois, à chaque variable définie en définition 3.2-2 est associé un exposant $1 \leq p \leq n_{paths}$ correspondant au numéro du chemin parmi les n_{paths}. Les contraintes énoncées en définition 3.2-3 se font maintenant $\forall\ p \in [\![1\,;\, n_{paths}]\!]$, à l'exception de celle concernant l'unicité des sens : en effet, si une lecture participe dans un sens donné dans un chemin, alors ni elle, ni son sens inverse-complémentaire ne peuvent participer dans

un autre chemin. Cela s'écrit :

$$\forall\ (v_f, v_r) \in R_i,\ \sum_{p=1}^{n_{paths}} s^p_{v_f} + i^p_{v_f} + t^p_{v_f} + s^p_{v_r} + i^p_{v_r} + t^p_{v_r} \leq 1$$

Postulat 3.2-2 : Fonction objective

On admet que la reconstitution de la région locale du génome associée à ce sous-graphe correspond à trouver les plus lourds chemins, selon les longueurs de chevauchements λ_{E_i}, dans ce graphe. La fonction objective associée est : $maximizer \min_{1 \leq p \leq n_{paths}} \left(\sum_{(u,v) \in E_i} \lambda_{uv} \times x^p_{uv} \right)$

3.2.3 Concaténation des chemins des sous-graphes

Définition 3.2-4 : Chemins élémentaires acceptés

Soit la liste de nœuds $Path^p_i,\ 1 \leq p \leq n_{paths}$ le p-ième plus lourd chemin élémentaire trouvé et accepté dans le sous-graphe OG_i.

Afin de procéder à la concaténation, trois ensembles de nœuds sont utilisés, nommés ensembles d'états :

Définition 3.2-5 : Ensembles d'états des nœuds

— S_i l'ensemble des nœuds qui peuvent être débutants d'un chemin

— I_i l'ensemble des nœuds qui ne peuvent être qu'intermédiaires d'un chemin

— T_i l'ensemble des nœuds qui peuvent être terminaux d'un chemin

La valeur de ces ensembles dépend de la position du sous-graphe OG_i dans l'ordre des sous-graphes à résoudre, *cf.* axiome 3.1-1 et annexe A.1, définition A.1-1.

Un tel découpage ensembliste (avec possible redondances lors de l'union), rend les contraintes ensembles d'états spécifiques, ce qui accélère la résolution des problèmes. Ainsi, les variables s, i, t définies en 3.2-2 ne sont plus définies sur l'ensemble entier V_i mais sur les ensembles S_i, I_i, T_i, à savoir :

$$s : S_i \to \{0\,;1\} \qquad i : S_i \cup I_i \cup T_i \to \{0\,;1\} \qquad t : T_i \to \{0\,;1\}$$

Une réécriture des contraintes s'impose alors. Elle ne sera toutefois pas précisée ici car relève davantage de réécritures techniques d'implémentation.

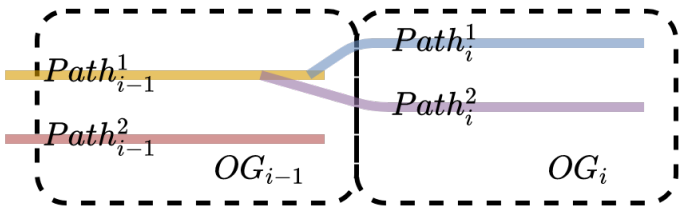


Figure 3.5 – Continuité des chemins entre les sous-graphes. Le débutant de plusieurs chemins dans le sous-graphe OG_i peuvent venir d'un même chemin dans le sous-graphe précédent OG_{i-1}. Toutefois, il ne peut y avoir de croisements entre deux chemins issus du sous graphe précédent qui fusionneraient en un seul chemin dans le sous-graphe en cours de résolution OG_i.

Enfin, pour chaque chemin $Path_i^p$ dans OG_i, connaissant l'origine du sommet débutant, c'est à dire de quel chemin $Path_{i-1}^q$ il venait, une simple concaténation s'opère à partir de ce sommet pivot. La suite du chemin $Path_{i-1}^q$ après le dernier sommet pivot est simplement effacée. Il découle de cette propriété qu'il existe un chemin entre le débutant d'un des chemins du premier sous-graphe OG_1 et le terminal d'un des chemins du dernier sous-graphe $OG_{n_{parts}}$ dans le graphe d'origine OG.

À présent, un grand nombre de nœuds participent dans les chemins parallèles des sous-graphes, *i.e.* un grand nombre de lectures ont été positionnées par rapport à un même axe de position *pos*. Un tel positionnement aide à la réalisation d'un consensus de ces séquences. En effet, la valeur des positions des lectures dépendant de la similarité (longueur de chevauchement) entre deux lectures consécutives dans les chemins élémentaires des sous-graphes, il est, dans un premier temps, admis que deux lectures proches en positions sont similaires en séquences.

3.3 Découpage de l'axe des positions

Afin de préparer des instances raisonnables en taille - selon la longueur d'un sous axe de position et de sa profondeur locale associée, et de profiter du positionnement des lectures, nous avons décidé de découper l'axe objectif de position *pos* en fenêtres que l'on nommera pour la suite *fenêtres* Ω. Il s'agira de réaliser un consensus dans chaque fenêtre Ω et de procéder à la fusion des consensus locaux. La partie présentée est actuellement une ébauche de recherche. Deux méthodes de réalisation de séquences consensus en local dans chaque fenêtre Ω seront mises en exergue.

La problématique soulevée ici est : comment découper d'une manière optimale cet axe de positions *pos* pour plus tard faciliter la fusion des séquences consensus de chaque fenêtre ? Nous proposons ici un découpage à l'aide d'une modélisation en un PLMNE à partir d'un concept simple : les coupures doivent s'opérer dans des zones où le consensus est le plus facilement déterminable, *i.e.* où l'on retrouve en forte abondance un même motif nucléotidique, que l'on nommera alors *ancre*.

3.3.1 Recherche d'ancres par zones τ pour découpage

Postulat 3.3-1 : Recherche d'ancres par décalage de zones τ

Soit $k \in \mathbb{N}^*$ la longueur d'un kmer, *i.e.* la taille d'une petite séquence nucléotidique. L'axe *pos* est divisé en zones τ de longueur $l_\tau \geq k$, et distantes de $\Delta_\tau \geq 1$ nucléotides. Dans chaque zone τ, les sous-séquences des lectures chevauchant la zone sont découpées en kmers, par décalage de un nucléotide. Un comptage de chaque kmer dans la zone est réalisé, et le kmer présent en plus grand nombre est appelé *ancre*.

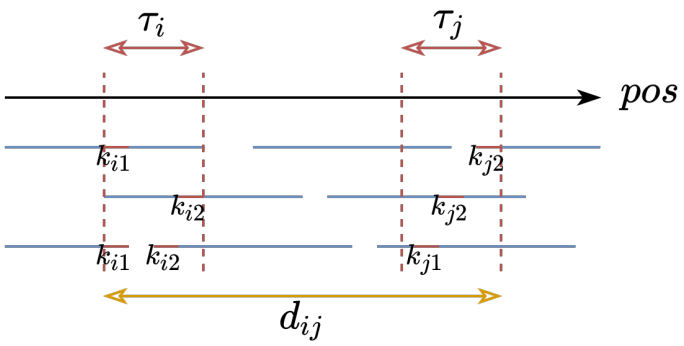


FIGURE 3.6 – Détermination d'ancres par approximations en zones τ de longueur l_τ. Les lectures (en bleues) ont été positionnées sur l'axe des positions *pos*. Les parties en rouge sur les lectures représentent des kmers. Deux kmers de même séquence dans une même zone τ sont considérés comme un même kmer sur le génome. d_{ij} représente la distance séparant deux zones τ_i et τ_j

La motivation d'un tel postulat vient de la problématique des erreurs de séquençage de type *insup*. En effet, une suppression ou une insertion de nucléotides dans les lectures par rapport à la région du génome qu'elles couvrent induit un décalage des séquences sur l'axe des positions *pos*. La notion de recherche du kmer de plus grande abondance dans une zone τ, *i.e.* d'une ancre, répond à cette problématique en proposant une recherche flexible sur la région des positions. Une recherche stricte correspondant à $l_\tau = k$.

3.3.2 Définition des bornes des fenêtres Ω

On souhaite à présent déterminer les bornes des fenêtres Ω, définies par des ancres. Cela est équivalent à chercher une suite de zones τ, qui se suivent sur l'axe *pos*, tel que l'on favorise l'abondance globale des ancres associées.

Définition 3.3-1 : Graphe de zones τ G_τ

Soit $G_\tau = (V, E, ab_V)$ le graphe représentant les liaisons possibles entre des zones τ, où V est l'ensemble des zones pondérées par l'abondance de leur ancre ab_V, et E l'ensemble des arcs qui sont les liaisons entre les zones, défini par :

$$(\tau_i, \tau_j) \in E \iff \begin{cases} i < j \\ \Omega_{inf} \leq d_{ij} \\ \Omega_{sup} \geq d_{ij} \end{cases} \quad \text{où } C \times k \leq \Omega_{inf} \leq \Omega_{sup},\ C \text{ une constante.}$$

Déterminer la suite de zones τ revient à déterminer un chemin dans ce graphe G_τ. Nous proposons une résolution par PLMNE pour ce problème, qui est un problème d'optimisation globale. Les variables sont similaires à celles énoncées en définition 3.2-2

Proposition 3.3-1 : Modèle τ - Fonction objective

Choisir les zones τ les plus abondantes est équivalent à la recherche du chemin le plus lourds, par conséquent, la fonction objective est : $maximizer \sum_{v \in V} ab_v \times (s_v + i_v + t_v)$

Preuve : la preuve de cette proposition est triviale sachant la modélisation et la nature du problème.

Les contraintes pour ce problème sont décrites de manière analogue à la définition des contraintes 3.2-3.

Maintenant, chaque zone τ_i participante au chemin possède deux coordonnées, τ_{i1} et τ_{i2}, sur l'axe *pos*. Deux zones τ qui se suivent dans le chemin bornent une fenêtre Ω.

3.3.3 Selection des séquences dans chaque fenêtre Ω

Définition 3.3-2 : Séquences contenues dans une fenêtre Ω

Soit une fenêtre Ω bornée par les zones τ_i et τ_j d'ancres respectives $anch_i$ et $anch_j$. Soit une lecture r chevauchant totalement ou en partie cette fenêtre. La sous séquence r_Ω de r alors retenue dans la fenêtre Ω est :

$$r_\Omega = \begin{cases} r[anch_i : anch_j] & anch_j \text{ comprise, si } r \text{ possède les deux ancres} \\ r[anch_i : \tau_{j2}] & \text{si } r \text{ ne possède que l'ancre } anch_i \\ r[\tau_{i1} : anch_j] & \text{si } r \text{ ne possède que l'ancre } anch_j \\ r[\tau_{i1} : \tau_{j2}] & \text{sinon} \end{cases}$$

Les $X_{thr} \in \mathbb{N}^*$ plus longues sous-séquences sont gardées.

Dans le cas de la deuxième méthode de consensus détaillée en sous-section 3.4.2, seules les longueurs de sous séquences supérieures à un certain ratio de la longueur de la fenêtre Ω sont gardées, *i.e* si $|\, r_\Omega \,| \geq \rho_\Omega \times l_\Omega,\ \rho_\Omega \in [0\,;1],\ l_\Omega = \tau_{j2} - \tau_{i1}$.

3.4 Realisation du consensus dans chaque fenêtre Ω

Chaque fenêtre Ω est à présent composée de sous-séquences r_Ω. Il faut ensuite produire une séquence nucléotidique unique appelée séquence consensus dans chacune des fenêtres. Deux méthodes

seront présentées : la première sera davantage détaillée, et bien qu'elle ne donne pour l'instant pas de résultats positifs, elle a l'avantage de s'appuyer sur le positionnement des lectures ; la deuxième est une première idée (car peu détaillée) de la mise en place d'un alignement multiple dans ce contexte.

3.4.1 Consensus par k-merisation des lectures avec PLMNE

Cette méthode est une construction d'une séquence consensus à partir de kmers chevauchant extraits des lectures r_Ω, en s'aidant de la positions de ces kmers sur l'axe *pos*. Deux kmers ne pourront se chevaucher que s'ils sont proches l'un de l'autre sur l'axe *pos*. Elle hérite d'une méthode de consensus par programmation dynamique dans le cadre du projet de stockage d'informations sur ADN[25] proposée par D. Lavenier. L'algorithme n'y étant pas décrit car est toujours en phase de développement. Toutefois, la méthode ne sera pas précisée car les résultats sont encore préliminaires, et continuera à être développée jusqu'à la fin du stage.

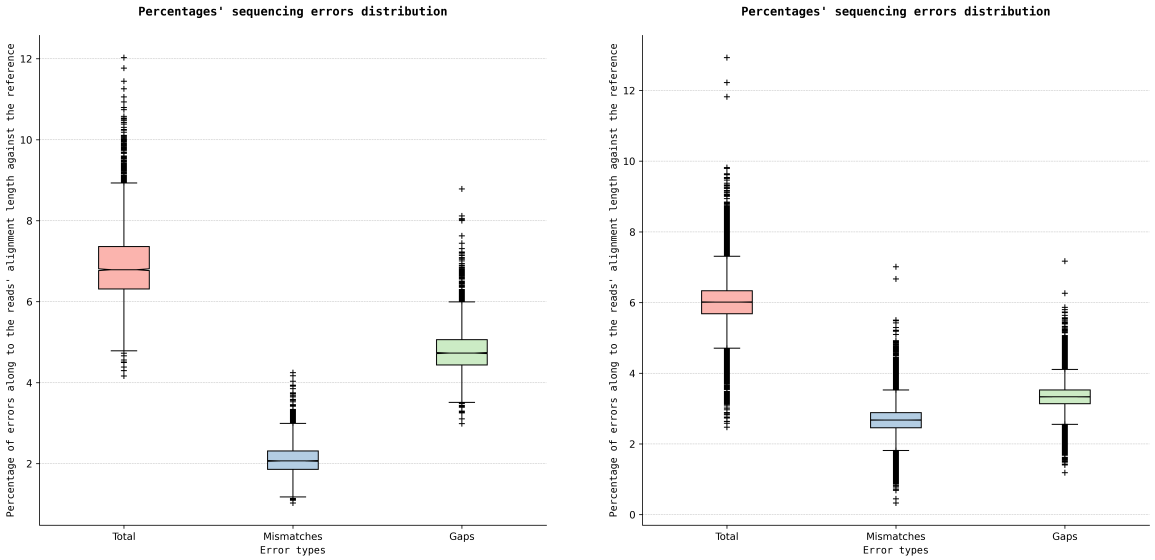
3.4.2 Alignements multiples

Construction en parallèle de séquences consensus : à cette fin, le programme d'alignement multiple de séquences MUSCLE est utilisé. Chaque alignement multiple ne prenant qu'un seul processus, la parallélisation s'effectue sur les fenêtre Ω *i.e.* une fenêtre vaut un processus. Les options utilisées sont **-diags -maxiters 4**, la première est un paramètre d'accélération et la deuxième limite le nombre d'itérations pour raffiner le résultat. Une fois les alignements obtenus (qui sont des séquences toutes de mêmes tailles, avec une lettre représentant les *insup*), une unique séquence est produite par vote majoritaire, avec pour règle que si la lettre majoritaire était un *insup*, alors elle n'apparaît pas dans la séquence consensus. Pour le moment, aucune méthode de fusion n'est proposée pour cette méthode de construction de séquence consensus.

Chapitre 4 : Résultats

4.1 Analyse préalable des données

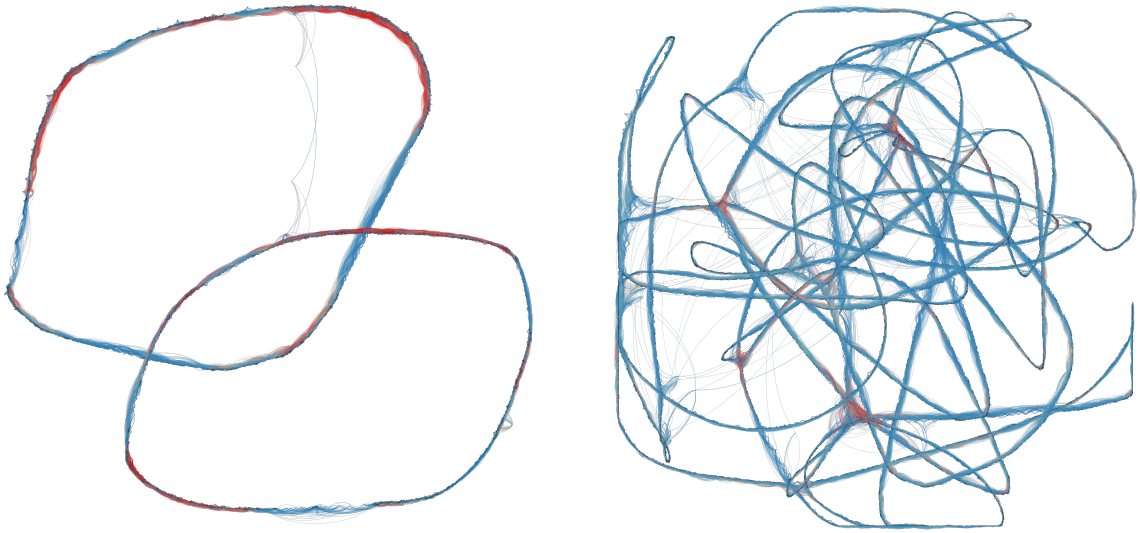
Afin d'illustrer les erreurs de séquençages sur lesquels la stratégie a été testée, à la fois sur le jeu de données réelles Sthe et un jeu de données simulé (Abau), les longues lectures ont été comparées contre leur génome de référence en les alignant avec Blast. Ensuite ont été extraits les pourcentages des types d'alignements par rapport à la longueur des alignements : identités (non indiqué ici), substitutions (*mismatches*), insertions et suppressions (regroupées en une catégorie *insup*, *gaps* ici).



(a) Sthe erreurs de séquençage : boîtes à moustaches de Notch

(b) Abau erreurs de séquençage : boîtes à moustaches de Notch

Figure 4.1 – Les erreurs de séquençage pour un jeu de données réelles (Sthe) et un jeu de données simulé (Abau). Le bleu et le vert sont respectivement associés aux pourcentage de substitutions et d'*insup*. Le rouge est le pourcentage d'erreur global. (4.1a) et (4.1b) : distribution en boîtes à moustaches de Notch des pourcentages d'erreurs de séquençages. Même si la différence s'observe davantage chez Sthe que chez Abau, les *insup* semblent plus nombreux que les substitutions. Le taux d'erreurs global (en rouge) médian avoisine les 7% et les 6% pour Sthe et Abau resp.



(a) Graphe de chevauchements pour Sthe (b) Graphe de chevauchements pour Abau

Figure 4.2 – Graphes de chevauchements pour les jeux de données Sthe et Abau, visualisés avec Gephi. Les arcs héritent de la couleur de leur nœud source pour un meilleur aperçu : rouge signifie que le nœud source à un degré (nombre d'arcs entrant et sortant) élevé contrairement à la couleur bleue.
(4.2a) : la circularité de ce génome bactérien est ici flagrante. Deux « anneaux » sont présents : ils correspondent aux deux brins ADN complémentaires l'un de l'autre. Une région répétée inverse-complémentaire existe donc dans ce génome pour que les deux brins soient simultanément représentés dans ce graphe. Ce graphe a été obtenu en ne gardant que les chevauchements au dessus de 19 346 nucléotides, comptabilisant ainsi 5 518 nœuds et 161 294 arcs.
(4.2b) : bien que ce graphe ne présente pas deux « anneaux » comme le précédent, les deux brins ADN sont présents. Cependant, on peut remarquer la conséquence directe des régions répétées dans le génome qui forment des boucles dans le graphe, formant alors des groupes de nœuds très denses (en rouge). Ces groupes induisent comme des carrefours peuvent induire en erreur concernant le choix des chemins, en passant d'une région génomique à une autre éloignée dans le génome. Ce graphe a été obtenu en ne gardant que les chevauchements au dessus de 5 203 nucléotides, comptabilisant ainsi 17 073 nœuds et 270 940 arcs.

4.2 Recherche de multi-chemins dans un graphe de chevauchements

Afin de trancher entre les deux méthodes, elles ont été réalisées sur un ensemble de 100 sous-graphes de chevauchements associés au jeu de données simulé Abau. Ce choix de jeu de test est basé sur une observation non décrite ici : le jeu de données Abau renvoyait des chevauchements entre des lectures qui ne faisaient pas parties du même brin ADN, et par conséquent, ce jeu de données est plus sensible aux erreurs de positionnements. Pour les deux méthodes, jusqu'à 4 chemins sont recherchés. Un temps limite d'exécution utilisateur de 10 secondes a été indiqué.

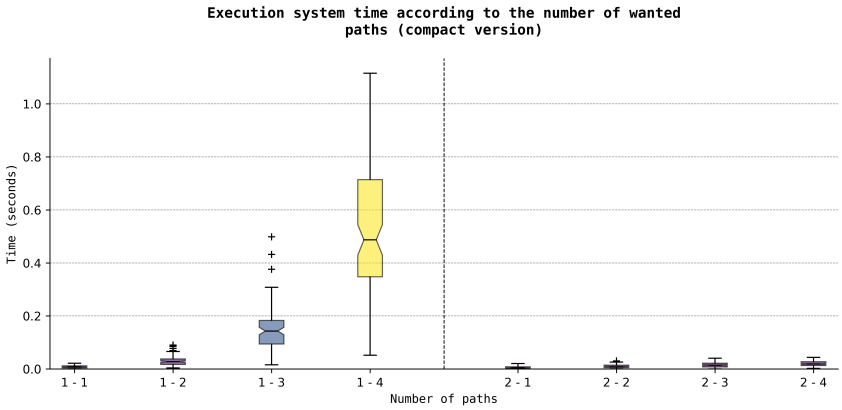


Figure 4.3 – Comparaison des temps d'exécution CPU entre la méthode parallèle (à gauche de la ligne en pointillée) et de la méthode séquentielle (à droite, chaque boîte à moustaches étant obtenue avec la somme des temps de recherche à l'itération courante et précédentes.). Dans cette figure et celles qui suivront, pour la méthode en parallèle, trouver k chemins se fait en une itération (noté 1-k sous les boîtes). Pour la méthode séquentielle, il faut k itérations (notées 2-i avec $1 \leq i \leq k$, $i.e.$ la recherche du i^{e} chemin).

Concernant les erreurs de positionnement, deux types d'erreur sont décrits : les erreurs fortes et les pseudo-erreurs. Les lectures possèdent un ensemble de coordonnées de *mapping* sur le génome de référence. Ainsi, on peut comparer les coordonnées retrouvées sur le génome de deux lectures se suivant dans le chemin. Si une lecture v est après une lecture u dans le chemin ($i.e.$ $x_{uv} = 1$), alors qu'en réalité v étant avant u d'au moins 100 nucléotides sur le génome de référence, alors on considère pour le couple de coordonnées qu'il y a une erreur. Si pour toutes les permutations de couple de coordonnées il y a une erreur, alors on considère qu'il y a une erreur forte. Sinon, s'il y a au moins une erreur, mais pas à tous les coups, on dit qu'il y a une pseudo-erreur.

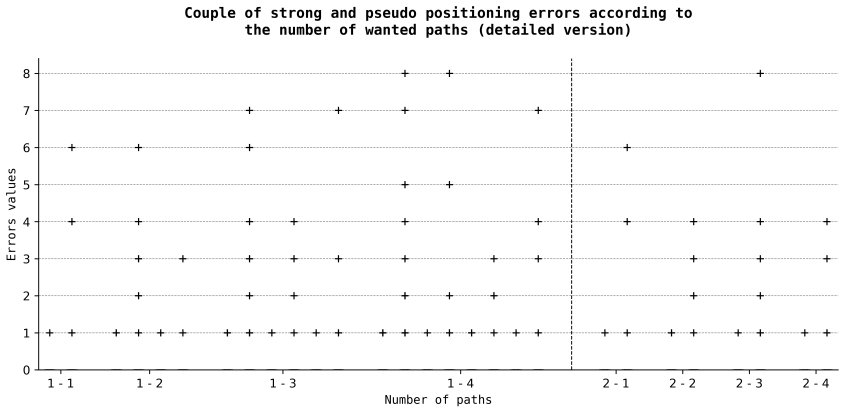


Figure 4.4 – Comparaison du nombre d'erreur des chemins (en détaillé pour la méthode parallèle). Pour chaque chemin est représenté le nombre d'erreurs fortes puis le nombre de pseudos erreurs. Il était attendu que le nombre de pseudo-erreurs soit supérieur au nombre d'erreurs fortes. Aussi, le nombre d'erreurs souvent égal à 0, les boîtes à moustaches sont écrasées sur l'axe des abscisses.

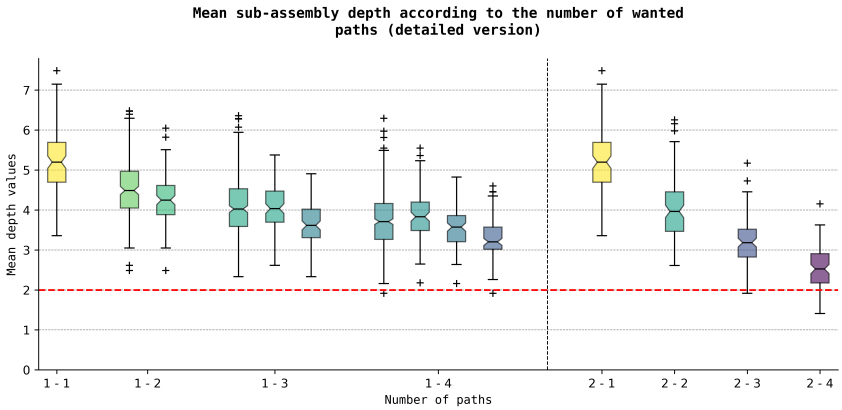


FIGURE 4.5 – Comparaison des ratios des chemins (en détaillé pour la méthode parallèle). Les ratios sont égaux à la somme des chevauchements choisis divisé par la longueur du chemin ($cf.$ postulat A.1-1). Cela a permit pour plus tard de choisir un ratio limite ρ arbitrairement égal à 2 (ligne horizontale en pointillée rouge).

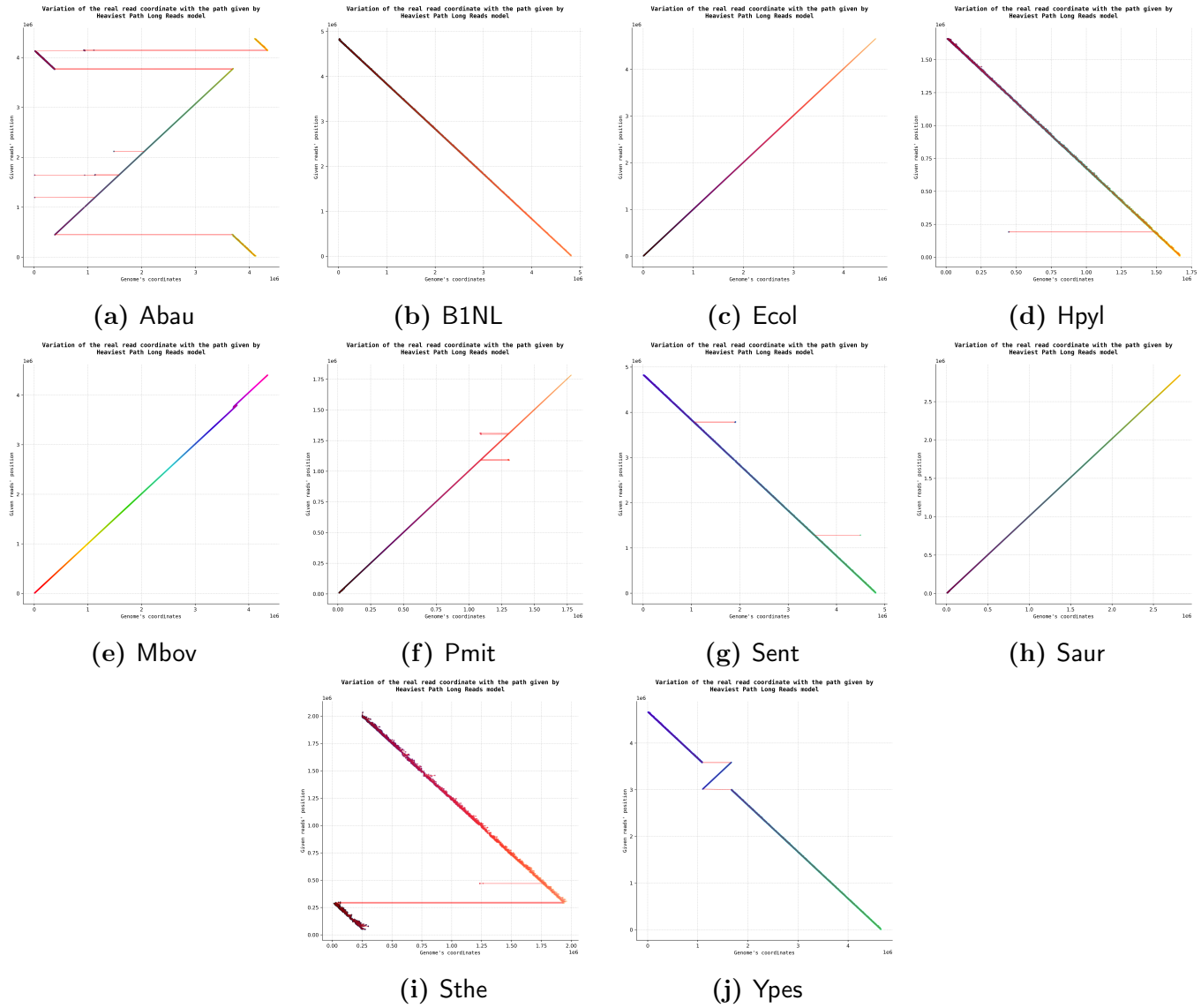
La différence de temps entre les deux méthodes est formelle : la méthode parallèle est bien plus lente que celle en séquentielle. Cela s'explique, pour la méthode parallèle, par des ensembles de chemins dont les valeurs de la fonction objective (le plus léger des plus lourds chemins, $cf.$ postulat 3.2-2) se ressemblent. Ainsi, atteindre une valeur objective du point de vue combinatoire peut s'avérer très lent. Ensuite, il n'est pas possible de conclure sur une quelconque différence sur le nombre d'erreurs entre les deux méthodes. Enfin, il est intéressant de noter que le ratio poids d'un chemin divisé par sa longueur semble plus haut pour la méthode en parallèle qu'en séquentielle. En effet, l'avantage de la recherche en parallèle fournit un optimale selon le chemin le plus léger, alors que la méthode séquentielle va favoriser les chemins les plus lourds que possibles pour les itérations précédentes. Un poids de chemins plus lourds signifie des régions mieux couvertes. Par conséquent, si l'on filtre selon un ratio limite (égal à 2 pour la suite), on risque de ne pas prendre les chemins après une certaine itération. Toutefois, on préférera moins de chemins mais qui sont plus lourds, et donc plus susceptibles d'être bon. Par conséquent, la méthode de recherche séquentielle a été sélectionnée.

4.2.1 Résultats de positionnement pour la méthode séquentielle

Les hauts comptages d'erreurs fortes pour Abau et Ypes (table 4.1) ne sont globalement pas liés à de mauvais positionnements locaux dans les sous-graphes mais correspondent à un ordre de résolution donné de ces sous-graphes qui est faux ($cf.$ sous-figures 4.6a et 4.6j). Cette observation remet en cause le point (ii) de l'axiome 3.1-1. Le cas de Sthe est particulier car les erreurs fortes semblent être liées à la circularité du génome, alors que la référence est considérée linéaire lors de l'attribution des coordonnées des lectures sur celle ci ($cf.$ figure 4.6i).

Bactéries	LCM	NN / NA	EF / PE	Temps CPU / réel	Pic RAM
Abau	5 203	17 073 / 270 940	2 747 / 21	743,93 / 527,57	5,90
B1NL	4 603	20 016 / 349 465	1 / 6	853,53 / 713,48	7,43
Ecol	5 400	17 988 / 284 246	0 / 0	722,23 / 509,68	5,84
Hpyl	5 258	6 487 / 103 403	0 / 15	213,85 / 133,83	2,40
Mbov	5 437	16 821 / 265 569	3 / 0	554,24 / 320,02	5,57
Pmit	5 757	6 649 / 100 343	0 / 70	223,42 / 170,72	2,23
Sent	5 619	18 270 / 277 722	11 / 2	732,63 / 515,26	5,68
Saur	5 427	10 932 / 170 683	0 / 0	318,14 / 180,30	3,70
Sthe	19 346	5 518 / 161 294	4 / 328	597,73 / 290,47	4,07
Ypes	5 273	18 213 / 277 602	1522 / 8	698,14 / 494,30	5,85

Table 4.1 – Bactéries : identifiants des jeux de données ; LCM : Longueur de Chevauchement Minimale entre deux lectures en nucléotides, qui correspond au 3^{e} quartile des longueurs de chevauchements ; NN / NA : nombre de nœuds / d'arcs totaux dans le graphe de chevauchements d'origine ; EF / PE : somme du nombre d'erreurs fortes / pseudo-erreurs dans chaque chemin ; Temps CPU / réel : temps en seconde ; Pic RAM : le maximum de mémoire RAM demandée en GB.



(a) Abau (b) B1NL (c) Ecol (d) Hpyl

(e) Mbov (f) Pmit (g) Sent (h) Saur

(i) Sthe (j) Ypes

FIGURE 4.6 – Les positions des lectures en fonction de leur coordonnée sur le génome de référence. sont reliés par segment rouge deux points consécutifs sur l'axe des positions. Une variation globalement croissante signifie que le positionnement s'est effectué sur le même brin ADN que celui de la référence, alors qu'une variation globalement décroissante signifie qu'il s'est effectué sur le brin inverse complémentaire.

4.3 Découpage de l'axe des positions *pos* en fenêtres Ω

Nous avons ensuite testé le découpage de l'axe des positions *pos* et la production de consensus sur le jeu de données Sthe. En effet, nous avons favorisé les tests sur ce jeu de données car il est issu d'un séquençage réel. Concernant la découpe de l'axe *pos*, nous avons choisis une taille d'ancre de 8, une taille de zones τ $l_\tau = 100$, distantes chacune de $\Delta_\tau = l_\tau$ (sans chevauchements des zones), et des tailles de fenêtres Ω comprises entre 1000 et 1500. Aussi, on ne gardera que des longueurs de lectures r_Ω dans une fenêtre que si elles dépassent 90% ($\rho_\Omega = 0,9$, *cf.* définition 3.3-2) de sa taille. Aucune limite X_{thr} de nombre de lectures r_Ω n'a été stipulée (*i.e.* $X_{thr} \to \infty$). Cet axe a été découpé en 2161

fenêtres Ω.

La découpe de l'axe *pos* pour Sthe a duré 344.15 sec. CPU (276.52 sec. temps réel) et a atteint en mémoire RAM 0.72 GB.

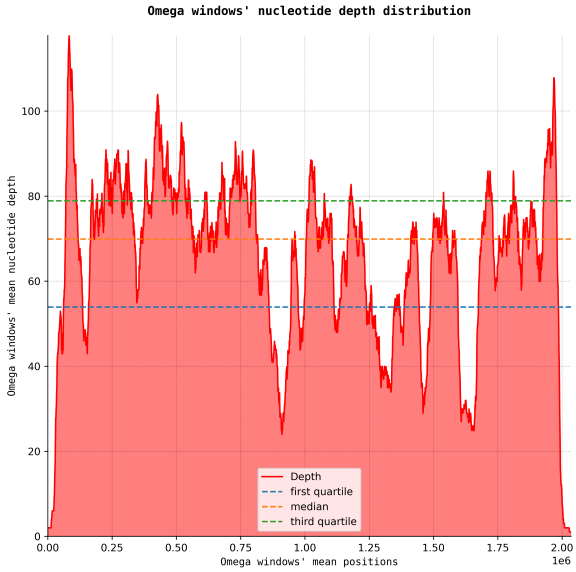
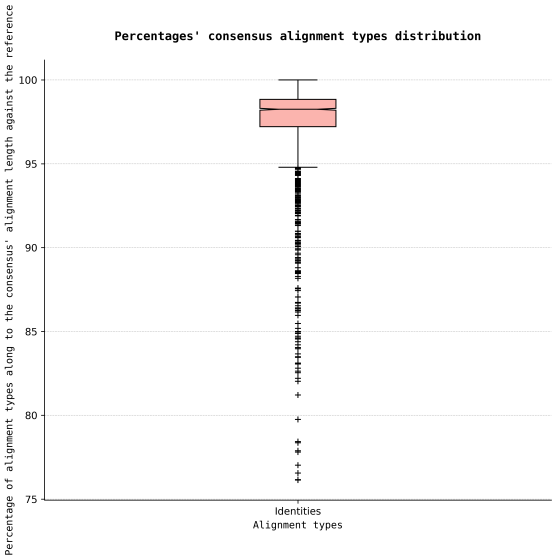


Figure 4.7 – Résultat de la coupure en fenêtres Ω pour le jeu de données Sthe. Est représenté la profondeur, *i.e.* la somme des longueurs des séquences r_Ω divisée par la taille de la fenêtre Ω, en fonction de la position de la fenêtre sur l'axe des positions. Les premiers, deuxième et troisième quartiles sont représentés par les lignes horizontales bleue, jaune et verte respectivement. Comme attendu, la profondeur est faible aux bords : la réponse est plus évidente pour la fin de l'axe des positions car elle n'hérite pas des chevauchements postérieurs ; au début de l'axe, cela peut s'expliquer par les toutes premières lectures, car n'ayant pas la même taille, certaines couvrent seules le début de l'axe, ce qui peut être un artefact.

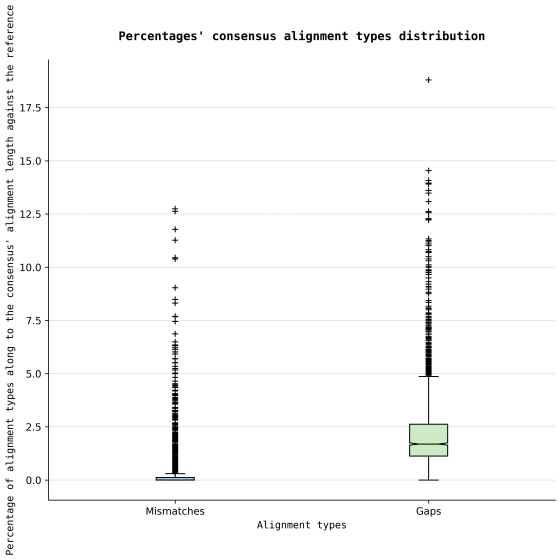
4.4 Consensus dans les fenêtres Ω

Une fois les séquences consensus créées, elles sont comparées avec BLAST contre le génome de référence. Les résultats présentés ici ne sont qu'un aperçu de ce que l'on pourrait attendre des méthodes. Malheureusement, seuls des résultats émanant de la méthode par l'alignement multiple seront montrés ici, car le consensus par k-merisation n'a pas donné de résultats positifs. En effet, les instances ont parfois été infaisables, très longues en temps et gourmandes en mémoire. Un trop grand nombre de paramètres doit être déterminé pour obtenir une solution, qui souvent renvoyait de mauvais résultats.

La production de toutes les séquences consensus a duré 82 604, 024 sec. CPU (11 063, 37 sec. temps réel en utilisant 8 cœurs en parallèle). Le pic de mémoire RAM ayant atteint 0.12 GB. Pour chaque fenêtre Ω, une séquence consensus est renvoyée. Chacune est comparée avec la référence. L'alignement gardé parmi les alignements donnés est celui qui compte le plus grand nombre d'identités. Les pourcentages présentés sont le nombre d'identités nucléique et d'erreurs sur l'alignement divisé par sa taille :

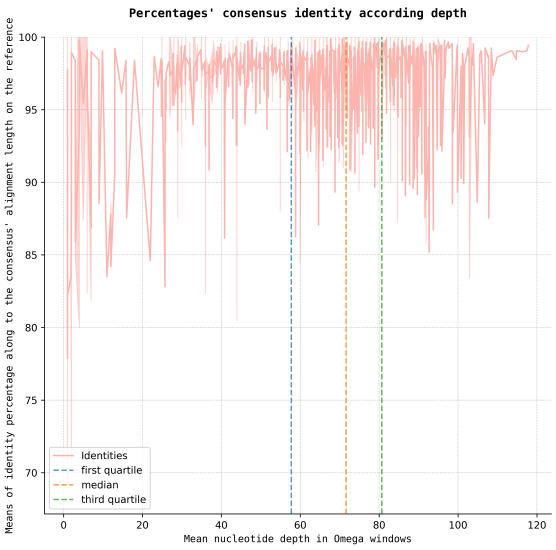


(a) Boîte à moustaches pour le pourcentage d'identités

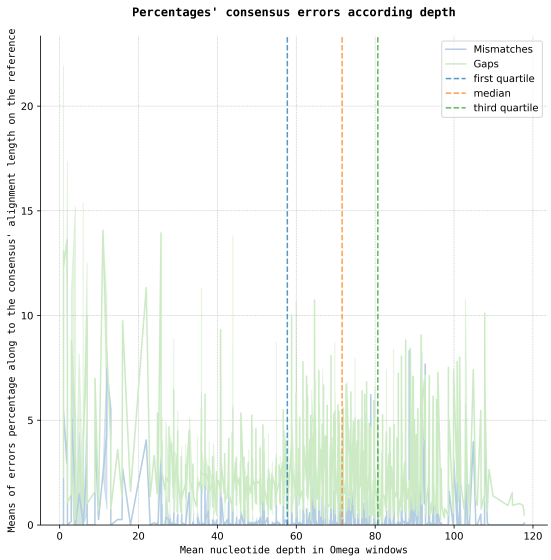


(b) Boîte à moustaches pour les pourcentages d'erreurs d'alignements

FIGURE 4.8 – Les distributions des identités et des erreurs d'alignements entre les séquences consensus et la référence. (4.8a) : le pourcentage médian avoisine les 97% mais quelques séquences ont un pourcentage d'identité faible (jusqu'à 76%). (4.8b) : assez critique, les erreurs de substitutions montent parfois à plus de 12% et les types $insup$ jusqu'à plus de 17% (les médianes respectives avoisinent les $0,5\%$ et les 2%).



(a) Distribution du pourcentage d'identité



(b) Distribution des pourcentages d'erreurs

FIGURE 4.9 – Les distributions d'identité (4.9a) et des erreurs d'alignements (4.9b) entre les séquences consensus et la référence en fonction de la profondeur des fenêtres Ω. Étonnamment, le pourcentage d'identité et d'erreurs ne semblent pas respectivement augmenter et baisser en fonction d'une plus grande profondeur.

Chapitre 5 : Discussion

Bien que les choix entre les deux méthodes de positionnement parallèle ou séquentielle nous semble justifié en raison de leur comparaison, le choix d'un ratio ρ limite et d'un pourcentage de longueur du premier chemin pour valider un nouveau chemin lors d'une nouvelle itération (*c.f.* postulat A.1-1) doivent être questionnés. Sont-ils de bonnes conditions de validation, et, le cas échéant, qu'en est-il du choix de leur valeur ? $\rho = 2$ se justifie par le souhait d'avoir une couverture minimale des chevauchements contre la longueur du chemin d'au moins égal à 2. Toutefois, les longueurs de chevauchements comptées ne le sont qu'entre deux lectures consécutives dans le chemin, ce qui ne prend pas en compte les chevauchements entre toutes les lectures de ce chemin. Le pourcentage limite $\chi = 0.9$ prévient des raccourcis dans le sous-graphe au cas où celui-ci couvrirait des régions répétées.

Alors qu'il semble que le positionnement se soit bien déroulé pour Sthe, notre idée qu'une plus grande profondeur globale de séquençage, et locale dans une fenêtre Ω, permettait d'obtenir une séquence consensus de meilleure qualité n'est ici pas vérifiée. Toutefois, ce n'est pas ce paradigme qu'il faut remettre en cause, mais le postulat 3.2-1 et sa généralisation pour k itérations. En effet, comme ce sont des chemins élémentaires qui sont construits en local, le positionnement d'une lecture s'effectue selon celui de la lecture précédente. Par récursion, il est donc fonction du positionnement de la première lecture du chemin inter sous-graphe de chevauchements, qui est égal à la concaténation d'un chemin élémentaire issu de chaque sous-graphes précédents. Par conséquent, en regardant la définition des attributs d'un chevauchement 3.1-1 et l'écriture des contraintes concernant la valeur des positions données définition 3.2-3, il en résulte un biais dans le positionnement, hérité des biais dans les sous-graphes précédents, et aggravé dans le sous-graphe en cours de résolution. Il s'ensuit donc un décalage des lectures entre les chemins déterminés à des itérations différentes pour un même sous-graphe. Ainsi, lors de la découpe en fenêtres Ω, les séquences r_Ω dans chaque fenêtre font parties de régions plus éloignées du génome les unes par rapport aux autres à mesure que l'on s'avance sur l'axe des positions *pos*.

Ce décalage devait être corrigé à travers l'usage des zones τ, zones de recherche flexible d'ancres lors de la découpe de l'axe *pos* en fenêtres Ω. De même, se pose la question du choix de leur longueur : une trop petite zone τ séparerait des kmers en des ancres différentes alors qu'ils couvraient le génome aux mêmes coordonnées, mais, de trop longues zones induisent un biais, en particulier lorsque des régions du génome possèdent des répétitions courtes en tandem - la distinction ne s'opérant alors pas.

Si une longueur d'au moins 1000 nucléotides a été choisie pour la taille des fenêtres Ω dans l'idée que l'alignement multiple corrigerait les décalages de positionnements énoncés plus haut (car on espère que ce décalage ne soit pas au plus de 500 nucléotides), de trop grandes tailles de fenêtres Ω ne permettent pas à la méthode de consensus par k-merisation d'aboutir. En effet, l'arbre combinatoire

des possibles explose en taille, et même la recherche d'une solution admissible - bien que non optimale, devient une tâche très longue à accomplir. Cet inconvenient sur la taille de l'arbre peut se réduire en diminuant la taille des fenêtres. Facilement réalisable, le décalage de positions est un biais fatal car cette méthode s'appuie sur un positionnement assez précis des kmers (de l'ordre de la taille d'une zone τ). On en revient alors au choix de la taille de ces zones.

Chapitre 6 : Conclusion et perspectives

Dans le souhait de s'assurer de la production d'une séquence consensus d'une meilleur qualité que possible à partir de longues lectures ayant un fort taux d'erreurs de séquençage en particulier de type *insup*, nous avons proposé une méthode de recherche multi-chemins élémentaires dans un graphe de chevauchements de lectures pour profiter de la profondeur de séquençage et de l'atteinte d'un optimal par la PLMNE. Ainsi, les chemins mis bout-à-bout dans l'ordre donné des sous-graphes de chevauchements permettent d'associer une position nucléotidique à chaque lecture utilisée dans les chemins. Cette association devait suivre celle qui à une lecture donne sa coordonné véritable sur le génome de référence. Cela semble être le cas pour la majorité des jeux de données testés.

Une fois les lectures placées sur un même axe de positions, celui ci est découpé en fenêtres et des séquences consensus sont produites pour chacune. Deux méthodes ont alors été abordées : une à partir d'une découpe en kmer des séquences présentes dans une fenêtre en s'appuyant sur les positions des lectures, l'autre par une méthode d'alignement multiple. Malheureusement pour la première, les biais induits dans l'attribution des positions des lectures n'ont pas permis d'obtenir de résultats positifs, et la deuxième méthode, bien qu'elle en a renvoyé, a souffert du décalage nucléotidique des séquences pour cause les *insup* et la méthode de positionnement.

Dans les deux cas, la méthode de recherche de chemins élémentaires dans un graphe de chevauchements, mais aussi la concaténation des chemins élémentaires de chaque sous-graphe dans l'ordre donné de résolution, n'est pas assez précise et souffre des conséquences des erreurs d'insertions et de suppressions dans les lectures. Un perfectionnement pour le positionnement pourrait-être une correction des positions *aposteriori* de la recherche de chemins, et ce, avec la PLMNE. Encore, il pourrait s'agir d'une redéfinition des contraintes de positionnements en ne donnant non pas qu'une position mais deux, résultant en un intervalle de positions pour chaque lecture.

Par ailleurs, les recherches autour de la production de consensus par k-merisation ne doivent pas être lâchées car pourraient au moins profiter à d'autres projets de recherche, à savoir celui cité à l'origine de la méthode. De même, sa précision (en choisissant des kmers à concaténer) reste intéressante. De l'autre côté, les recherches concernant l'alignement multiple pour produire un consensus est une idée attrayante car elle peut s'abstraire de petits décalages nucléotidiques induits par le positionnement. De plus, ce qui a été présenté n'est que le début de cette recherche qui nous semble prometteuse.

Enfin, des améliorations d'implémentations s'imposent pour réduire le temps de calcul et diminuer la mémoire RAM consommée par des ré-écritures d'objets sous formes matricielle, et ainsi s'éviter l'usage de structures lourdes sur les graphes avec NetworkX. Aussi, pour une vision plus lointaine, nous souhaitons nous détacher d'AMPL, programme commercial payant, en se tournant vers des alternatives gratuites, de préférence Open Source, telles que Pyomo ou JuMP.

Bibliographie

[1] Avershina, E. & Rudi, K. Dominant short repeated sequences in bacterial genomes. *Genomics* **105**, 175–181 (2015). URL http://www.sciencedirect.com/science/article/pii/S0888754314002821.

[2] Shen, R. *et al.* High-throughput SNP genotyping on universal bead arrays. *Mutation Research* **573**, 70–82 (2005).

[3] Jain, M., Olsen, H. E., Paten, B. & Akeson, M. The Oxford Nanopore MinION : delivery of nanopore sequencing to the genomics community. *Genome Biology* **17**, 239 (2016). URL https://doi.org/10.1186/s13059-016-1103-0.

[4] Eid, J. *et al.* Real-Time DNA Sequencing from Single Polymerase Molecules. *Science* **323**, 133–138 (2009). URL https://science.sciencemag.org/content/323/5910/133. Publisher : American Association for the Advancement of Science Section : Report.

[5] Koren, S. *et al.* Canu : scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research* **27**, 722–736 (2017). URL http://genome.cshlp.org/content/27/5/722. Company : Cold Spring Harbor Laboratory Press Distributor : Cold Spring Harbor Laboratory Press Institution : Cold Spring Harbor Laboratory Press Label : Cold Spring Harbor Laboratory Press Publisher : Cold Spring Harbor Lab.

[6] Ruan, J. & Li, H. Fast and accurate long-read assembly with wtdbg2. *Nature Methods* **17**, 155–158 (2020). URL https://www.nature.com/articles/s41592-019-0669-3.

[7] Kolmogorov, M., Yuan, J., Lin, Y. & Pevzner, P. A. Assembly of long, error-prone reads using repeat graphs. *Nature Biotechnology* **37**, 540–546 (2019). URL https://www.nature.com/articles/s41587-019-0072-8.

[8] Gay, D. M. The AMPL Modeling Language : An Aid to Formulating and Solving Optimization Problems. In Al-Baali, M., Grandinetti, L. & Purnama, A. (eds.) *Numerical Analysis and Optimization*, vol. 134, 95–116 (Springer International Publishing, Cham, 2015). URL http://link.springer.com/10.1007/978-3-319-17689-5_5.

[9] Hagberg, A. A., Schult, D. A. & Swart, P. J. Exploring Network Structure, Dynamics, and Function using NetworkX. In Varoquaux, G., Vaught, T. & Millman, J. (eds.) *Proceedings of the 7th Python in Science Conference*, 11 – 15 (Pasadena, CA USA, 2008).

[10] Hunter, J. D. Matplotlib : A 2D Graphics Environment. *Computing in Science Engineering* **9**, 90–95 (2007).

[11] Walt, S. v. d., Colbert, S. C. & Varoquaux, G. The NumPy Array : A Structure for Efficient Numerical Computation. *Computing in Science & Engineering* **13**, 22–30 (2011). URL https://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37.

[12] Virtanen, P. *et al.* SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261–272 (2020). URL http://arxiv.org/abs/1907.10121. ArXiv : 1907.10121.

[13] Bastian, M., Heymann, S. & Jacomy, M. Gephi : an open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media* (2009).

[14] Land, A. H. & Doig, A. G. An Automatic Method of Solving Discrete Programming Problems. *Econometrica* **28**, 497 (1960). URL https://www.jstor.org/stable/1910129?origin=crossref.

[15] Rothberg, E., Bixby, R. & Gu, Z. Gurobi Optimization - The State-of-the-Art Mathematical Programming Solver. URL http://www.gurobi.com/index.

[16] Li, H. Minimap2 : pairwise alignment for nucleotide sequences. *Bioinformatics* **34**, 3094–3100 (2018). URL https://academic.oup.com/bioinformatics/article/34/18/3094/4994778.

[17] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology* **215**, 403–410 (1990).

[18] Karypis, G. & Kumar, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* **20**, 359–392 (1998). URL http://dx.doi.org/10.1137/S1064827595287997.

[19] Epain, V., Lavenier, D., Djidjev, H. & Andonov, R. *De novo long reads assembly using integer linear programming.* other, Université de Rennes 1 [UR1] (2019). URL https://hal.inria.fr/hal-02413832.

[20] Li, Y. *et al.* DeepSimulator : a deep simulator for Nanopore sequencing. *Bioinformatics* **34**, 2899–2908 (2018). URL https://academic.oup.com/bioinformatics/article/34/17/2899/4962495.

[21] Edgar, R. C. MUSCLE : a multiple sequence alignment method with reduced time and space complexity. *BMC bioinformatics* **5**, 113 (2004).

[22] Wang, L. & Jiang, T. On the complexity of multiple sequence alignment. *Journal of Computational Biology : A Journal of Computational Molecular Cell Biology* **1**, 337–348 (1994).

[23] François, S., Andonov, R., Djidjev, H. & Lavenier, D. Global Optimization Methods for Genome Scaffolding (2016). URL https://hal.inria.fr/hal-01385665.

[24] Miller, C. E., Tucker, A. W. & Zemlin, R. A. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM* **7**, 326–329 (1960). URL https://doi.org/10.1145/321043.321046.

[25] Lavenier, D., Roux, E., Conde-Canencia, L. & Hamoum, B. *Advanced Coding Schemes for DNA-Based Data Storage Using Nanopore Sequencing Technologies* (2019). URL https://hal.archives-ouvertes.fr/hal-02400656. Published : Journées CominLabs 2019.

Chapitre A : Annexe

A.1 Méthodes de positionnements

Postulat A.1-1 : Critère d'arrêt de recherche de chemin

Soit L_i^1 la longueur nucléotidique associée au premier chemin $Path_i^1$ trouvé à l'itération 1. $\forall\, k \geq 2$, le chemin $Path_i^k$, trouvé à l'itération k, est accepté si, et seulement si :

$$\left(\frac{\sum\limits_{(u,v)\in E_i} \lambda_{uv} \times x_{uv}}{L_i^k} \geq \rho \right) \wedge (L_i^k \geq \chi \times L_i^1),\ \text{où } \rho \in \mathbb{R},\ \chi \in [0\,;1] \text{ sont des constantes.}$$

Définition A.1-1 : Ensembles d'états des nœuds

- S_i l'ensemble des nœuds qui peuvent être débutants d'un chemin
- I_i l'ensemble des nœuds qui ne peuvent être qu'intermédiaires d'un chemin
- T_i l'ensemble des nœuds qui peuvent être terminaux d'un chemin

La valeur de ces ensembles dépend de la position du sous-graphe OG_i dans l'ordre des sous-graphes à résoudre, *cf.* axiome 3.1-1. Soit $OG_{i-1} = (V_{i-1}, E_{i-1}, l_{V_{i-1}}, \lambda_{E_{i-1}}, g_{E_{i-1}})$ et $OG_{i+1} = (V_{i+1}, E_{i+1}, l_{V_{i+1}}, \lambda_{E_{i+1}}, g_{E_{i+1}})$, respectivement, quand ils existent, les graphes antérieur et postérieur au sous-graphe OG_i dans l'ordre de résolution des sous-graphes.

(i) Si OG_i est le premier sous-graphe :
- $S_i = V_i$
- $I_i = \varnothing$
- $T_i = \{u \in V_i \mid \exists\, v \in V_{i+1} \wedge (u,v) \in E\}$

(ii) Si OG_i est un sous-graphe au milieu de l'ordre de résolution :
- $S_i = \{u \in V_{i-1} \mid \exists\, v \in V_i,\ (u,v) \in E \wedge \exists p \in [\![1\,;\, n_{paths}]\!],\ u \in Path_{i-1}^p\}$ (*)
- $I_i = V_i \setminus (S_i \cup T_i)$
- $T_i = \{u \in V_i \mid \exists\, v \in V_{i+1},\ (u,v) \in E\}$

(iii) Sinon, OG_i est le dernier sous-graphe :
- $S_i = \{u \in V_{i-1} \mid \exists\, v \in V_i,\ (u,v) \in E \wedge \exists p \in [\![1\,;\, n_{paths}]\!],\ u \in Path_{i-1}^p\}$ (*)
- $I_i = \varnothing$
- $T_i = V_i$

(*) Dans les cas (ii) et (iii), $V_i \leftarrow V_i \cup S_i$ et $E_i \leftarrow E_i \cup \{(u,v) \in E \mid u \in S_i \wedge v \in V_i\}$ *i.e.* le sous-graphe courant OG_i hérite des nœuds du sous-graphes précédent OG_{i-1} qui font la liaison entre les chemins précédents et courants.

Master de Bioinformatique de l'Université de Rennes 1 - 2019 / 2020

Victor EPAIN

Amélioration du positionnement de fragments ADN pour réalisation de séquences consensus dans le contexte de l'assemblage *de novo* de longues lectures

L'analyse *in silico* d'une molécule d'ADN requiert son séquençage en fragments appelés lectures puis leur assemblage. Les technologies de séquençage produisant de longues lectures offrent l'avantage de surpasser les problèmes de régions répétées dans les génomes qui sont alors plus facilement couvertes dans leur entièreté, mais produisent des lectures avec un fort taux d'erreurs qui se réfèrent davantage à des insertions ou des suppressions de nucléotides, nommées *insup*. L'assemblage *de novo* est un assemblage qui ne s'aide d'aucune référence. Bien que des méthodes d'assemblage de longues lectures ont déjà été proposés, par l'usage de graphes de De Bruijn ou par correction successives des lectures par exemple, nous proposons une stratégie qui s'opère en deux étapes : positionner le maximum de lectures sur un axe de positions, puis produire une séquence consensus en s'aidant des positions déterminées. À ces fins, nous proposons une modélisation du problème du positionnement avec la programmation mathématique linéaire mixte en nombres entiers (PLMNE), et présentons de premières idées pour la production de séquences consensus avec la PLMNE et l'alignement multiples de séquences à partir de ce positionnement. La finalité de cette stratégie étant de formaliser la problématique d'assemblage de génome, nous l'avons structurée selon la méthode mathématique, ce qui permet de cibler les choix méthodologiques précisément afin de réduire l'usage d'heuristiques. Enfin nous avons testé cette stratégie sur des génomes bactériens. Bien que les résultats de positionnements soient globalement très positifs, ceux pour le consensus le sont moins mais n'enlèvent pas aux méthodes leur potentialités.

graphe de chevauchements, problème du plus lourd chemin, programmation mathématique, graphe de décalage de kmers, alignement multiple

DNA fragments positioning improvement to realise consensus sequences in the *de novo* long reads assembly context

DNA molecular *in silico* analysis requires sequencing it in fragments called reads, and then assembling them. Today's, long reads sequencing technologies offer the possibility to overcome genome's repeated regions issues with being entirely covered, but product high errors rate data with sequencing errors, like nucleotides insertions or deletions, called *indels*. *De novo* assembly is an assembly without using a reference. Although some assemblers already exist according to several methods - as using De Bruijn graphs or by correcting iteratively the reads for example, we propose a two steps strategy : first, we attribute to the maximum of reads a position on a same positions axis and then we product a consensus sequence thanks to the positioning. At these aims, we propose a modelling for the positioning issue with the mixed integer linear programming (MILP), and we present first ideas for the consensus sequences production with MILP too and multiple sequences alignment from the positioning. As the final aim of this strategy is to formalize the genome assembly issue, we structured it according the mathematical method, that permits to target methodological choices precisely, and then reducing the heuristic uses. Finally, we tested the strategy with bacteria genomes. Despite the fact that positioning results show positive ones, the consensus results are less positive but don't remove the potentiality of the associated methods.

overlaps graph, heaviest path problem, mathematical programming, kmers shifting graph, multiple alignment